



Attorney Docket No. GB920000078US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application

Applicant(s): Anthony J. O'Dowd
Docket No.: GB920000078US1
Serial No.: 09/682,520
Filing Date: September 13, 2001
Group: 2193
Examiner: Trenton J. Roche

I hereby certify that this paper is being deposited on this date with the U.S. Postal Service as first class mail addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Signature:  Date: October 5, 2005

Title: Tracing the Execution Path of a Computer Program

APPEAL BRIEF

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313

Sir:

Applicant (hereinafter referred to as "Appellant") hereby appeals the final rejection of claims 1-15 of the above-referenced application.

REAL PARTY IN INTEREST

The present application is assigned to International Business Machines Corp., as evidenced by an assignment recorded September 13, 2001 in the U.S. Patent and Trademark Office at Reel 11953, Frame 257. The assignee, International Business Machines Corp., is the real party in interest.

RELATED APPEALS AND INTERFERENCES

There are no known related appeals and interferences.

10/07/2005 HDESTA1 00000052 500510 09682520

01 FC:1402 500.00 DA

STATUS OF CLAIMS

Claims 1-15 are pending in the present application. Claims 1, 13 and 14 stand rejected under 35 U.S.C. §102(e). Claims 2-12 and 15 stand rejected under 35 U.S.C. §103(a). Claims 1-15 are appealed.

STATUS OF AMENDMENTS

There have been no amendments filed subsequent to the final rejection.

SUMMARY OF CLAIMED SUBJECT MATTER

The present invention, by way of example, as recited in independent claim 1, provides a method for tracing the execution path of a computer program comprising at least one module including a plurality of instructions, wherein at least one of the instructions is a branch instruction. The method comprises the steps of identifying each branch instruction, evaluating each branch instruction to be one of true and false, and responsive to an evaluation of true, pushing a unique identifier into a predefined area of storage, wherein the unique identifier is associated with the instructions executed as a result of the evaluation of true (Specification, page 15, lines 1-8).

Independent claim 13 recites, *inter alia*, a pusher, responsive to an evaluation of true, for pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true (Specification, page 16, lines 14-22).

Independent claim 14 recites, *inter alia*, the step of instrumenting said instructions associated with an evaluation of true with a signature instruction, wherein said signature instruction causes a unique identifier to be pushed into a predefined area of storage upon execution of said true instructions at run-time (Specification, page 16, line 23, through page 17, line 4).

Independent claim 15 recites, *inter alia*, a pusher for instrumenting said instructions associated with an evaluation of true with a signature instruction, wherein said signature instruction causes a unique identifier to be pushed into a predefined area of storage upon execution of said true instructions at run-time (Specification, page 17, lines 5-17).

In accordance with an illustrative embodiment depicted in FIGs. 4 and 5, upon execution of the program, a small, fixed size area called a signature area is defined. The signature area may contain up to a certain number of signature points. Each signature point comprises a unique 4 bit identifier. This identifier is, according to this illustrative embodiment, used to indicate the execution path or flow followed through the program. According to this illustrative embodiment, signature points are added to the signature area. For example, case statement 1 is evaluated to TRUE such that 1 is pushed into the signature area 300. Execution then jumps to case statement 4, case statement 2, case statement 3, case statement 1, and finally to case statement 2. Accordingly, the corresponding identifiers are pushed into the signature area (1, 4, 2, 3, 1, 2). These numbers indicate which set of instructions have been executed at run-time and in what order. The signature information provides valuable insight into the behavior of the program. Should the program fail or behave erroneously, then the signature points can be used in subsequent problem diagnostics (Specification, page 9, line 20, through page 12, line 6).

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

- (1) Claims 1, 13 and 14 are rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Patent No. 6,173,395 to Wisor et al. (hereinafter “Wisor”).
- (2) Claims 2 and 3 are rejected under 35 U.S.C. §103(a) as being unpatentable over Wisor.
- (3) Claims 4-12 and 15 are rejected under 35 U.S.C. §103(a) as being unpatentable over Wisor in view of U.S. Patent No. 6,353,924 to Ayers et al (hereinafter “Ayers”).

ARGUMENT

- (1) Claims 1, 13 and 14 are rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Patent No. 6,173,395 to Wisor et al. (hereinafter “Wisor”).

Regarding the §102(e) rejection of claims 1, 13 and 14, Appellant asserts that Wisor fails to teach or suggest all of the limitations in said claims for at least the reasons presented below.

It is well-established law that a claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference.

Verdegaal Bros. v. Union Oil Co. of California, 814 F.2d 628, 631, 2 U.S.P.Q.2d 1051, 1053 (Fed. Cir. 1987). Appellant asserts that the rejection based on Wisor does not meet this basic legal requirement, as will be explained below.

By way of example, the invention of independent claim 1 recites a method for tracing the execution path of a computer program comprising at least one module including a plurality of instructions, at least one of said instructions being a branch instruction, the method comprising the steps of identifying each branch instruction, evaluating each branch instruction to be one of true and false, and responsive to an evaluation of true, pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true. Claim 13 recites, *inter alia*, a pusher, responsive to an evaluation of true, for pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true. Claim 14 recites, *inter alia*, the step of instrumenting said instructions associated with an evaluation of true with a signature instruction, wherein said signature instruction causes a unique identifier to be pushed into a predefined area of storage upon execution of said true instructions at run-time.

In accordance with an illustrative embodiment explained in the present specification, at paragraphs 48 and 49, upon execution of the program, a small, fixed size area called a signature area is defined. The signature area may contain up to a certain number of signature points. Each signature point comprises a unique 4 bit identifier. This identifier is, according to this illustrative embodiment, used to indicate the execution path or flow followed through the program. According to this illustrative embodiment, signature points are added to the signature area. For example, as explained at paragraph 57 of the present specification with respect to FIG. 5, case statement 1 is evaluated to TRUE such that 1 is pushed into the signature area 300. Execution then jumps to case statement 4, case statement 2, case statement 3, case statement 1, and finally to case statement 2. Accordingly, the corresponding identifiers are pushed into the signature area (1, 4, 2, 3, 1, 2). These numbers indicate which set of instructions have been executed at run-time and in what order. The signature information provides valuable insight into the behavior of the program. Should the

program fail or behave erroneously, then the signature points can be used in subsequent problem diagnostics (paragraph 58).

The Examiner suggests that Wisor discloses all the features of the claimed invention. Appellant respectfully asserts that this is incorrect. Wisor does not disclose “pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true,” as recited in claim 1; “a pusher, responsive to an evaluation of true, for pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true,” as recited in claim 13; or “instrumenting said instructions associated with an evaluation of true with a signature instruction, wherein said signature instruction causes a unique identifier to be pushed into a predefined area of storage upon execution of said true instructions at run-time,” as recited in claim 14.

To the extent that the Examiner suggests that Wisor discloses such claimed features at column 3, lines 11-21, this is clearly incorrect. Column 3, lines 11-21, of Wisor state:

When a test program is executed, a trace record is generated and stored in the BTHB [branch trace history buffer]. The trace record consists of full entries and bitmap entries. The full entries are generated for unconditional branches and other significant trace events. (“Full” entries, as used herein, are those entries which each correspond to a single branch or trace event.) The full entries contain information relating to the target addresses of the branches. The bitmap entries are generated for a series of conditional branches and contain individual bits which represent the taken or not-taken status of the branches.

Nothing in this passage from Wisor, nor any passage from Wisor, teaches or suggests “pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true,” as recited in claim 1; “a pusher, responsive to an evaluation of true, for pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true,” as recited in claim 13; or “instrumenting said instructions associated with an evaluation of true with a signature instruction, wherein said signature instruction causes a unique

identifier to be pushed into a predefined area of storage upon execution of said true instructions at run-time,” as recited in claim 14.

To the extent that the Examiner suggests that the bitmap entries of Wisor are equivalent to the unique identifier of the claimed invention, Appellant respectfully points out that such bitmap entries are expressly described as containing “individual bits which represent the taken or not-taken status of the branches.” Thus, there is nothing unique about the bits since they merely represent whether a branch is taken or not taken.

Despite the above remarks (first offered in Appellant’s response dated August 19, 2004), the final Office Action contends that Wisor at column 7, lines 16-19, provides further support for the claim rejections. Column 7, lines 16-19, state:

One embodiment of the invention uses a 32-bit bitmap to record conditional branch information. The bitmap is filled from the highest order bit to the lowest. The 32 bits of the bitmap occupy address field 51 of the BTHB entry (the low order 32 bits of the 64-bit entry.) The 16 bits of selector field 52 are not used in bitmap entries. The bitmap entries do, however, include tag information 54 to identify them as bitmap entries, and they include the same miscellaneous information 53 as the other entries

However, column 6, lines 22-46, explains what is meant by “tag information 54” and “miscellaneous information 53”:

FIG. 2 shows that each BTHB entry 50 comprises 64 bits. The high-order 4 bits (63:60) are a tag 54 which identifies the type of branch or event which corresponds to the entry. For example, a hexadecimal 0 may indicate a synchronization entry, while a hexadecimal 2 may indicate a jump or a call. The low order 32 bits (31:0) are used as an address field 51 for unconditional branches. Although this field is referred to as an address field, it can be used to identify the logical address of the next sequential instruction (or target instruction), to store a value entered in one of the microprocessor registers, or to represent a sequence of conditional branches which are taken or not taken, depending on the type of BTHB entry. If a series of conditional branches is indicated, the next higher 16 bits (47:32) are not used. If the low order 32 bits indicate a logical address, the next higher 16 bits identify a selector 52 for the code segment in which the target address is located. (The logical address consists of the selector and an offset which is specified in the low-order 32 bits.) The remaining bits (59:48) are used for other information 53, such as whether paging or protection are enabled, whether virtual or SMM modes are enabled, what the default

operating mode is and whether tracing should be suspended on recording of the entry. This remaining information is referred to herein as “miscellaneous” entry information.

Thus, despite the Examiner’s assertion (at page 9 of the final Office Action) that “the bitmap entries are ‘unique’ in that they contain individual bits, as well as relevant information such as tag information and miscellaneous information to uniquely identify the entries as bitmap entries and associate the entries with branches of a program,” it is clear that any information associated with a bitmap entry in Wisor is nothing more than “individual bits which represent the taken or not-taken status of the branches,” “a tag which identifies the type of branch or event [unconditional or conditional] which corresponds to the entry,” and miscellaneous information “such as whether paging or protection are enabled, whether virtual or SMM modes are enabled, what the default operating mode is and whether tracing should be suspended on recording of the entry.”

Thus, it is clear that two bitmap entries referring to “taken” conditional branches in Wisor may be the same and, therefore, not unique since both conditional branches would have the same bit set to a logic one to indicate it is a conditional branch that is “taken” (as opposed to “not taken”), the tag would be the same (since they are both conditional branches), and both branches may have the same miscellaneous information associated therewith (e.g., such as whether paging or protection are enabled, whether virtual or SMM modes are enabled, what the default operating mode is and whether tracing should be suspended on recording of the entry).

The fact that Wisor does not teach or suggest a “unique identifier” as in the claimed invention is further supported by the description at column 3, line 57, through column 4, line 9, of Wisor:

If a conditional branch is encountered, the debug software reads the next bit in the corresponding bitmap entry. If the previous trace event was not a conditional branch, the next entry in the BTHB should be a bitmap entry. Each bitmap entry, however, may contain information corresponding to a number of conditional branches. The encountered branch corresponds to the first bit in the bitmap. If the first bit is a 1, the branch was taken. The debug software will determine the next instruction from the conditional branch instruction itself. If the first bit is a 0, the branch was not taken and the debug software can simply continue with the next instruction in program order. If the trace event which preceded the conditional branch was another conditional branch, then the debug software had already read a bitmap to determine whether that branch had been taken. If less than all the bits in the

bitmap have been used to determine whether conditional branches have been taken, the next bit is used to determine whether the current branch was taken. If all of the bits have been used, the next entry (also a bitmap entry) will be used for the current branch.

Clearly, the reason why the “debug software” in Wisor must look at a previous bitmap entry and/or a next bitmap entry is because the entries are not unique. This problem is overcome by the claimed invention.

Therefore, to reiterate, nothing in Wisor teaches or suggests that the bitmap entry or creation of a bitmap entry may involve: “pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true,” as recited in claim 1; “a pusher, responsive to an evaluation of true, for pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true,” as recited in claim 13; or “instrumenting said instructions associated with an evaluation of true with a signature instruction, wherein said signature instruction causes a unique identifier to be pushed into a predefined area of storage upon execution of said true instructions at run-time,” as recited in claim 14.

Further, while Wisor does not teach or suggest the “unique identifiers” of the claimed invention, Wisor also fails to teach or suggest “evaluating each branch instruction to be one of true and false, and responsive to an evaluation of true, pushing a unique identifier into a predefined area of storage,” as in the claimed invention. That is, nothing in Wisor teaches or suggests pushing a unique identifier into a predefined area of storage when a branch instruction evaluation is one of true and false.

Lastly, the final Office Action suggests that “the individual bits are themselves unique, in that their organization is uniquely organize [sic] according to the invention of Wisor to represent branch information and save storage space.” However, as pointed out above, two bitmap entries referring to “taken” conditional branches in Wisor may be the same and, therefore, not unique . Thus, again, the Wisor trace approach does not meet the requirements of the claimed invention.

The Advisory Action dated June 6, 2005 maintains the same line of reasoning by asserting that “entries are unique as the entries do not overlap or replace one another; they have separate

addresses in memory and in the BTHB.” Whether or not this is true, the value (identifier) that is stored in accordance with the entry is not unique, as explained above when both entries represent a branch not taken. Therefore, there is no unique identifier that is being pushed in Wisor.

For at least these reasons, Appellant asserts that independent claims 1, 13 and 14 are patentable over Wisor.

(2) Claims 2 and 3 are rejected under 35 U.S.C. §103(a) as being unpatentable over Wisor.

Regarding the §103(a) rejection of claims 2 and 3, Appellants assert that such claims, which depend from claim 1, are patentable over Wisor for at least the reasons given above with respect to claim 1.

Furthermore, despite the fact that Official Notice has been taken regarding the well-known nature of “volatile memory” and “nonvolatile memory,” Appellants respectfully assert that the rejection is still deficient for at least the reasons given above with respect to claim 1.

Still further, regarding the §103(a) rejections of dependent claims 2 and 3, Appellant asserts that such claims are patentable for at least the above reasons, and because such claims recite patentable subject matter in their own right.

(3) Claims 4-12 and 15 are rejected under 35 U.S.C. §103(a) as being unpatentable over Wisor in view of U.S. Patent No. 6,353,924 to Ayers et al (hereinafter “Ayers”).

Regarding the §103(a) rejection of claims 4-12, Appellants assert that such claims, which depend from claim 1, are patentable over Wisor for at least the reasons given above with respect to claim 1.

Regarding the §103(a) rejection of claim 15, Appellants assert that Ayers fails to remedy the above-mentioned deficiencies of Wisor, and thus is patentable over the Wisor/Ayers combination for at least the above reasons. That is, both Wisor and Ayers fail to disclose “a pusher for instrumenting said instructions associated with an evaluation of true with a signature instruction, wherein said signature instruction causes a unique identifier to be pushed into a predefined area of storage upon execution of said true instructions at run-time,” as recited in claim 15. That is, among

other deficiencies, there is no disclosure in Wisor or Ayers of a unique identifier being pushed with respect to a true result in an evaluation of an instruction.

Furthermore, regarding the §103(a) rejections of dependent claims 4-12, Appellant asserts that such claims are patentable for at least the above reasons, and because such claims recite patentable subject matter in their own right.

By way of example, neither Wisor nor Ayers maintain a distinction between a “storage area” and a “file,” as in claim 4. Further, the Office Action fails to address the distinction. As illustratively described at page 6, lines 9-12, of the present specification, “[p]referably, . . . the contents of the storage area is output to . . . a file or trace log . . . at a predetermined point in time.” Then, as claims 5-8 further define, content outputting from the storage area to the file may be accomplished in a variety of ways. Wisor and Ayers are silent to these features.

Furthermore, neither Wisor nor Ayers disclose the features of claims 9-12. As illustratively explained at page 7, lines 16-20, of the present specification, “[a]ccording to the preferred embodiment, the storage area stores the position at which the last unique identifier was written . . . [t]hus it is possible to determine the number of unique identifiers that have been overwritten in the . . . storage area, before being output to the trace log . . . [i]f this number is excessively large, the size of the storage area is preferably increased.” These features are one embodiment of the features recited in claims 9-12.

The final Office Action refers to a wraparound feature, a branch counter being incremented, and a conditional branch check in Wisor, and a general note in Ayers that a user may increase a buffer limit, in support of the rejection of claims 9-12. However, these features in Wisor and Ayers are not the same as the features recited in claims 9-12. To the extent that the Examiner is still asserting that a “unique identifier” is a “bit entry” in Wisor, it is not clear how the conventional wraparound feature described in Wisor enables one to “determine the number of unique identifiers that have been overwritten in the storage area, before being output to the file (e.g., trace log),” as claimed.

Attorney Docket No. GB920000078US1

In view of the above, Appellant believes that claims 1-15 are in condition for allowance, and respectfully requests favorable reconsideration and withdrawal of the various rejections.

Respectfully submitted,



William E. Lewis
Attorney for Appellant(s)
Reg. No. 39,274
Ryan, Mason & Lewis, LLP
90 Forest Avenue
Locust Valley, NY 11560
(516) 759-2946

Date: October 5, 2005

APPENDIX

1. A method for tracing the execution path of a computer program comprising at least one module including a plurality of instructions, at least one of said instructions being a branch instruction, the method comprising the steps of:
identifying each branch instruction;
evaluating each branch instruction to be one of true and false; and
responsive to an evaluation of true, pushing a unique identifier into a predefined area of storage, wherein said unique identifier is associated with the instructions executed as a result of said evaluation of true.
2. The method of claim 1, further comprising the step of:
providing said predefined area of storage with volatile memory.
3. The method of claim 1, further comprising the step of:
providing said predefined area of storage with non-volatile memory.
4. The method of claim 1, further comprising the step of:
outputting the contents of said storage area to a file at a predetermined point in time.
5. The method of claim 4, further comprising the step of:
outputting trace information to said file upon exit from said at least one module.
6. The method of claim 5, further comprising the step of outputting the contents of said storage area at the same time as said exit trace information.
7. The method of claim 4, wherein the step of outputting the contents of said storage area further comprises the step of:
determining whether said storage area is full; and

responsive to a positive determination, outputting said contents to said file.

8. The method of claim 4, wherein the step of outputting the contents of said storage area further comprises the step of:

determining whether a failure has occurred within said program; and
responsive to a positive determination, outputting said contents to said file.

9. The method of claim 4, wherein the step of pushing a unique identifier into a predefined area of storage further comprises the steps of:

determining whether said predefined area of storage is full; and
overwriting the first unique identifier in said storage area.

10. The method of claim 9, further comprising the step of:

writing the position of the most recent unique identifier to be written out to said storage area to said storage area.

11. The method of claim 10, further comprising the step of using said position to determine the number of unique identifiers that have been overwritten prior to being written out to said file.

12. The method of claim 11, further comprising the step of:

responsive to determining that a large number of unique identifiers have been overwritten, increasing the size of said predefined area of storage.

13. An apparatus for tracing the execution path of a computer program comprising at least one module including a plurality of instructions, at least one of said instructions being a branch instruction, said apparatus comprising:

an identifier for identifying each branch instruction;

an evaluator for evaluating each branch instruction to be one of true and false; and
a pusher, responsive to an evaluation of true, for pushing a unique identifier into a
predefined area of storage, wherein said unique identifier is associated with the instructions
executed as a result of said evaluation of true.

14. A method for instrumenting a computer program comprising at least one module
including a plurality of instructions, at least one of said instructions being a branch instruction,
each branch instruction being evaluated to be one of true and false at run-time, with at least one
signature instruction for indicating the execution path of said program at run-time, the method
comprising the steps of:

identifying each branch instruction;

identifying the instructions associated with an evaluation of true at run-time;

instrumenting said instructions associated with an evaluation of true with a signature
instruction, wherein said signature instruction causes a unique identifier to be pushed into a
predefined area of storage upon execution of said true instruction at run-time.

15. A compiler for instrumenting a computer program comprising at least one module
including a plurality of instructions, at least one of said instructions being a branch instruction,
each branch instruction being evaluated to be one of true and false at runtime, with at least one
signature instruction for indicating the execution path of said program at run-time, said compiler
comprising:

a first identifier for identifying each branch Instruction;

a second identifier for identifying the instructions associated with an evaluation of true at
run-time;

a pusher for instrumenting said instructions associated with an evaluation of true with a
signature instruction, wherein said signature instruction causes a unique identifier to be pushed
into a predefined area of storage upon execution of said true instructions at run-time.